

IN THE CLAIMS:

1-19 (Canceled)

20-35. (Canceled)

1 **36. (Previously Presented)** A method for an original operating system (a
2 host OS) in a mobile device that supports a memory protection mechanism to run
3 another operating system (a guest OS) within the same memory space of said
4 host OS while preserving the current state of said host OS in memory throughout
5 the execution of said guest OS, comprising the steps of:

6 said mobile device running said host OS;

7 said host OS starting said guest OS through a launcher;

8 said launcher going through said memory protection mechanism to mark
9 memory blocks currently used by said host OS as protected from said guest OS;

10 said launcher launching said guest OS;

11 said guest OS running, accessing only memory blocks that have not been
12 marked as protected so that said memory blocks marked as protected are
13 preserved throughout the execution of said guest OS;

14 said guest OS finishing running through an exit-code;

15 said exit-code restoring the state of said host OS by reverting said
16 protected memory blocks; and

17 said host OS resuming its operation.

1 **37. (Previously Presented)** The method of claim 36, wherein said memory
2 protection mechanism is achieved through a memory management unit (MMU)
3 that allows or disallows a program to access particular memory addresses.

1 **38. (Previously Presented)** The method of claim 36, wherein said launcher
2 going through said memory protection mechanism to mark memory blocks

3 currently used by said host OS as protected from said guest OS further
4 comprises the steps of:
5 moving memory blocks currently used by said host OS whose address
6 range is needed by said guest OS to a free memory location in the mobile
7 device; and
8 marking said free memory blocks as protected;
9 and wherein restoring the state of said host OS by reverting said protected
10 memory blocks further comprises the steps of:
11 restoring said free memory blocks to said memory blocks whose address
12 range is needed by said guest OS; and
13 reverting said free memory blocks from protected.

1 39. (Previously Presented) The method of claim 36, further comprising
2 starting a second guest OS from said guest OS, wherein starting the second
3 guest OS comprises the steps of:
4 said guest OS acting as a host OS to the second guest OS;
5 said second guest OS acting as a guest OS; and
6 repeating the steps in claim 36 to start said second guest OS from said
7 guest OS.

1 40. (Previously Presented) A method for an original operating system (a
2 host OS) of a mobile device to start another operating system (a guest OS) while
3 keeping the running state and data of said host OS in memory throughout the
4 execution of said guest OS, comprising the steps of:
5 said mobile device running said host OS;
6 said host OS starting said guest OS through a launcher;
7 said launcher moving memory blocks in lower address space that are
8 currently used by said host OS to free memory blocks in upper address space,
9 and eventually preserving current state and data of said host OS to the upper
10 address space through the following steps:

11 for each of said memory blocks in lower address space currently
12 used by said host OS, finding a free memory block in the upper address space;
13 and
14 moving said used memory block in lower address space to said
15 free memory block in upper address space;
16 said launcher identifying a memory address location where memory
17 addresses above said location contains host OS data and memory addresses
18 below said memory address location are free to be used by said guest OS;
19 said launcher launching said guest OS by passing said memory address
20 location as a reduced memory size to said guest OS;
21 said guest OS running in said reduced in size memory space and leaving
22 memory space higher than said memory location untouched;
23 said guest OS finishing running through an exit-code;
24 said exit-code restoring the state and data of said host OS by reverting
25 each of said memory blocks in said lower address space from each of said free
26 memory blocks in said upper address space; and
27 said host OS resuming operation.

1 41. (Previously Presented) The method of claim 40, wherein said launcher
2 launching said guest OS by passing memory address location as a reduced
3 memory size to said guest OS further comprises:

4 a memory detection module of said guest OS that uses said reduced in
5 size memory space as the total available memory of the mobile device instead of
6 the real memory size of the mobile device.

1 42. (Previously Presented) The method of claim 40, wherein said launcher
2 launching said guest OS by passing memory address location as a reduced
3 memory size to said guest OS further comprises:

4 said launcher modifying a system register in said mobile device to report
5 said reduced memory size as total available memory to said guest OS.

1 43. (Previously Presented) The method of claim 40, wherein memories of
2 said mobile device are divided into multiple memory banks, and said launcher
3 launching said guest OS by passing memory address location as a reduced
4 memory size to said guest OS further comprises:

5 generating said reduced memory size by disabling one or more of said
6 memory banks to make them no longer available to said guest OS.

1 44. (Previously Presented) The method of claim 40, wherein input output
2 (IO) states and registers of said mobile device within said host OS are preserved
3 and later restored into said free memory block in upper address space.

1 45. (Previously Presented) A method for an original operating system (a
2 host OS) of a mobile device to start another operating system (a guest OS) within
3 the same memory space of said host OS while keeping the running state of said
4 host OS throughout the execution of said guest OS in place, comprising the
5 steps of:

6 said mobile device running said host OS;

7 said host OS starting said guest OS through a launcher;

8 said launcher launching said guest OS, passing a list of memory
9 addresses of currently used memory of said host OS to a memory device driver
10 in said guest OS during initialization of said guest OS;

11 said memory device driver of said guest OS claiming said list of memory
12 addresses and keeping them from being modified by any other part of said guest
13 OS during the execution of said guest OS;

14 said guest OS running in the same memory space of said host OS, with
15 memories used by said host OS being claimed and protected by said memory
16 device driver;

17 said guest OS finishing running through an exit-code;

18 said exit-code restoring the state of said host OS by releasing said list of
19 memory addresses from said device driver; and

20 said host OS resuming operation.

1 46. (Previously Presented) The method of claim 45, wherein said
2 list of memory addresses used by host OS includes memory addresses only in
3 dynamic runtime memory of said host OS and not memories.

1 47. (Previously Presented) The method of claim 45, wherein said
2 list of memory addresses used by host OS also includes the current input output
3 (IO) states and current registers of the mobile device.

1 48. (Canceled)

1 49. (Canceled)

1 50. (Previously Presented) A method of packaging an image of a guest
2 OS inside a native application of a host OS to allow in-place execution of the said
3 guest OS image in order to reduce memory usage where the format of said
4 native application contains multiple un-continuous data chunks in memory space,
5 comprising the steps of:

6 compiling an image of said guest OS into multiple code segments;

7 appending each of said multiple code segments with a jump table
8 containing multiple jump addresses to be used to point to others of the code
9 segments;

10 for each of a plurality of inter-segment jump instructions in each of said
11 code segments, linking each of said plurality of inter-segment jump instructions to
12 point to an entry of said jump table;

13 preparing a native application for said host OS with a startup code and
14 multiple data chunks;

15 each of said multiple data chunks wrapping each of said code segments of
16 said guest OS plus each of said jump tables;

17 said host OS launching said guest OS by running said native application;

18 said startup code in said native application looping through entries of each
19 of said jump table to fill in the current memory address of each corresponding
20 code segment;
21 each of said code segments executing a jump instruction to invoke codes
22 in the other code segments;
23 said executed jump instruction getting a real address of said other code
24 segment from said jump tables;
25 said executed jump instruction successfully jumping to the other code
26 segments wrapped by a corresponding data chunk; and
27 wherein said image of guest OS executes in place without the need to
28 extract all the code segments from said native application and rearrange them in
29 a continuous memory location.

1 51. (Previously Presented) The method of claim 50, wherein said
2 native application in said host OS is a database file with multiple data records or
3 a Palm PDB file with multiple Palm DB records.

1 52. (Previously Presented) The method of claim 50, wherein said native
2 application in said host OS is a Palm PDB file with multiple Palm DB records and
3 said host OS is a Palm OS.

1 53. (Previously Presented) The method of claim 50, wherein said image of
2 said guest OS code requires one sequential memory address, said native
3 application of said host OS contains multiple data chunks that have a maximum
4 chunk size, and compiling an image of said guest OS into multiple code
5 segments further comprises the steps of:
6 compiling said image of said guest OS into one code segment with
7 sequential memory addresses;
8 re-arranging instructions in said image of said guest OS to reserve spaces
9 to be used for said jump tables on each of a plurality of chunk-size boundaries by
10 inserting jump instructions to bypass those spaces; and

11 splitting said one code segment into multiple code segments on each of
12 said plurality of chunk-size boundaries.

1 54. (Previously Presented) The method of claim 50, wherein the size of
2 said guest OS image exceeds the maximum size limit of said native application
3 of said host OS, further comprising the steps of:
4 splitting said image of said guest OS into multiple pieces;
5 compiling each of said multiple pieces of said image of said guest OS into
6 multiple code segments; and
7 packaging each of said multiple pieces of said image of guest OS into a
8 native application of said host OS.